

Programmation Orientée Objet - Licence TIS CM8/9

Lancelot PECQUET

Lancelot.Pecquet@math.univ-poitiers.fr



Poitiers, le 13/03/2006

Rappel sur la séance précédente

La fois précédente, nous avons vu :

- ④ UML
- ② conception OO
- ③ motifs de conception (*design patterns*)
- ④ projet

Aujourd'hui, nous voyons :

- ① Fichiers
- ② Sérialisation
- ③ Threads
- ④ Applets

Sortie standard

Fichier Main.java

```
1 class Main{
2   public static void main(String[] argv){
3     System.out.println("HelloWorld");
4   }
5 }
```

Sortie (console)

```
1 $ java Main
2 HelloWorld
```

Entrée standard

Fichier Main.java

```
1 import java.io.*;
2
3 class Main{
4   public static void main(String[] argv){
5     BufferedReader console = new BufferedReader(new InputStreamReader(System.in));
6     System.out.print("vous_vous_prenommez?_");
7     String nom = "";
8     try{nom = console.readLine();}catch(IOException e){e.printStackTrace();}
9     System.out.println("Bonjour_" + nom);
10  }
11 }
```

Sortie (console)

```
1 $ java Main
2 Vous vous prenommez? Toto
3 Bonjour Toto
```

Écriture dans un fichier texte

Fichier Main.java

```
1 import java.io.*;
2
3 class Main{
4     public static void main(String[] argv){
5         File f = new File("toto.txt");
6         PrintWriter out = null; FileWriter fw = null;
7         try{fw = new FileWriter(f);}catch(IOException e){e.printStackTrace();}
8         out = new PrintWriter(fw); out.println("Coucou"); out.close();
9     }
10 }
```

Sortie (console)

```
1 $ java Main
2 $ cat toto.txt
3 Coucou
```

Lecture d'un fichier texte

Fichier Main.java

```
1 import java.io.*;
2 class Main{
3     public static void main(String[] argv){
4         FileReader fr = null;
5         try{fr = new FileReader("toto.txt");}catch(IOException e){e.printStackTrace();}
6         BufferedReader br = new BufferedReader(fr); String ligne = null;
7         try{
8             while((ligne = br.readLine()) != null){System.out.println(ligne);}
9             br.close();
10        }catch(IOException e){e.printStackTrace();}
11    }
12 }
```

Sortie (console)

```
1 $ java Main
2 Coucou
```

Méthodes de la classe File

méthode Java	spécification
<code>boolean exists()</code>	true ssi le fichier/répertoire existe
<code>boolean canRead()</code>	true ssi il est possible de lire dans le fichier
<code>boolean canWrite()</code>	true ssi il est possible d'écrire dans le fichier
<code>void delete()</code>	efface le fichier
<code>String getAbsolutePath()</code>	le chemin absolu du fichier/répertoire
<code>String getName()</code>	le chemin absolu du fichier/répertoire
<code>String getParent()</code>	le chemin absolu du répertoire parent
<code>boolean isDirectory()</code>	true ssi c'est un répertoire
<code>boolean isFile()</code>	true ssi c'est un fichier
<code>long lastModified()</code>	date de dernière modification
<code>long length()</code>	taille
<code>long list()</code>	liste des fichiers du répertoire

Questions ?



Principe de la sérialisation

Définition

La **sérialisation** transforme un objet volatile en objet persistant qui peut être stocké sur le disque ou envoyé sur le réseau puis récupéré à l'identique (indépendant de l'OS).

- implémentation (récursive) de l'interface `Serializable` (sinon : `NotSerializableException`)
- permet de tester l'égalité (`lent`)
- champs non-sérialisés : `transient`
- redéfinition possible de `readObject()` et `writeObject()`
- chiffrement possible. . .

Cas d'une classe personne

Fichier `personne.java`

```
1 class personne implements Serializable{
2     private String nom;
3     private String prenom;
4     private int taille;
5     public final String getNom(){return nom;}
6     public final String getPrenom(){return prenom;}
7     public final int getTaille(){return taille;}
8     public final void setNom(String nom){this.nom = nom;}
9     public final void setPrenom(String prenom){this.prenom = prenom;}
10    public final void setTaille(int taille){this.taille = taille;}
11    public personne(String nom, String prenom, int taille){
12        setNom(nom); setPrenom(prenom); setTaille(taille);
13    }
14    public String toString(){
15        return "nom:~" + getNom() + ",_prenom:~" + getPrenom()
16            + ",_taille:~" + getTaille() + "cm";
17    }
18    ...}
```

Sérialisation

Sérialiseur dans personne.java

```
1 import java.io.*;
2
3 class personne implements Serializable{
4     ...
5     // Sérialiseur (nom non consacré):
6     public void serialize(String F){
7         try {
8             FileOutputStream fichier = new FileOutputStream(F);
9             ObjectOutputStream oos = new ObjectOutputStream(fichier);
10            oos.writeObject(this);
11            oos.flush(); oos.close();
12        }catch (java.io.IOException e){e.printStackTrace();}
13    }
14
15    ...}
```

Désérialisation

Désérialiseur static dans personne.java

```
1 import java.io.*;
2
3 class personne implements Serializable{
4     ...
5     // Désérialiseur (nom non consacré):
6     public static personne deserialize(String nom.fichier){
7         personne p = null;
8         try{
9             FileInputStream fichier = new FileInputStream(nom.fichier);
10            ObjectInputStream ois = new ObjectInputStream(fichier);
11            p = (personne) ois.readObject();
12        }catch (java.io.IOException e){e.printStackTrace();}
13        }catch (ClassNotFoundException e){e.printStackTrace();}
14        return p;
15    }
16    ...}
```

Exemple de sérialisation/désérialisation

Fichier Main.java

```
1 class Main{
2     public static void main(String[] argv){
3         personne p = new personne("Doe", "John", 170); System.out.println(p);
4         p.serialize("pDoe.ser");
5         personne q = personne.deserialize("pDoe.ser"); System.out.println(q);
6         q.serialize("qDoe.ser");
7     }
8     ...}
```

Sortie (console)

```
1 $ java Main
2 nom: Doe, prenom: John, taille: 170cm
3 nom: Doe, prenom: John, taille: 170cm
4 $ diff pDoe.ser qDoe.ser
5 $
```

Questions ?



Le parallélisme, qu'est-ce que c'est ?

- plusieurs tâches sont réalisées simultanément, **en parallèle** :
 - traitement de texte ;
 - lecteur mp3 ;
 - réception d'un e-mail.
- multitâche :
 - plusieurs processus se déroulent en parallèle ;
 - typiquement, 1 programme = 1 processus ;
- *multithread* :
 - chaque processus est composé de « processus légers » (*light process*) ;
 - chacun suit son propre fil (*thread*) ;
 - typiquement, 1 processus = n threads.

Processus vs. threads

Il existe plusieurs formes de parallélisme, en particulier :

- à mémoire distribuée (processus) :
 - chaque processus travaille indépendamment des autres ;
 - chaque processus a une zone mémoire propre ;
 - la création d'un processus se fait par copie d'un autre processus (`fork()`) ;
- à mémoire partagée (*thread*) :
 - chaque thread travaille indépendamment (il suit son *fil*) ;
 - les threads partagent la même zone mémoire ;
 - la création d'un thread se fait librement ; il peut être démarré ou suspendu à la demande.

Quel est le rapport avec la POO ?

Classe (Thread) standard :

- création (constructeur);
- démarrage (start());
- suspension (wait());
- réveil (notify());
- destruction (finalize());
- passage en section critique (synchronized).

d'où :

- clarification des programmes;
- gain de temps de développement;
- portabilité.

Syntaxe Java pour utiliser un thread

Définition d'un thread

```
1 class un_thread extends Thread{
2   public void run(){ // redefinition
3     Thread.sleep(1000); // dort pendant 1s (attente passive)
4     System.out.println("coucou");
5   }
6 }
```

Utilisation d'un thread

```
1 public static void main(){...
2   un_thread T = new un_thread();
3   T.start(); // naissance de T: invoque run()
4   T.wait(); // suspend T
5   T.notify(); // reveille T
6   notifyAll() // reveille tous les threads
7   ...}
```

Exclusion mutuelle : méthodes synchronized

- variables communes aux différents threads \implies gestion des accès concurrents ;
- une méthode est **critique** lorsqu'elle utilise des variables partagées \implies déclaration synchronized ;
- deux méthodes synchronized ne peuvent s'exécuter simultanément.

Le dîner chinois des philosophes de Dijkstra

La situation

- 5 philosophes sont autour d'une table ronde ;
- certains pensent, tandis que d'autres mangent ;
- entre chaque assiette : une seule baguette (5 en tout) ;
- il faut 2 baguettes pour manger ;
- il n'y a pas de problème d'hygiène. . .

Le problème

- un max. de philosophes (2) doit manger simultanément ;
- aucun philosophe ne doit mourir de faim.

Résolution du problème avec un moniteur

Fichier Main.java

```
1 class Main{
2   public static void main(String[] args){
3     Monitor M = new Monitor(); // le gestionnaire de taches
4     Philosophe[] convives = new Philosophe[5]; // tableau de philosophes
5     for(int i=0; i<5; i++){
6       convives[i] = new Philosophe(i,M); // creation du convive i
7       convives[i].setName("Le_Philosophe_" + i); // nommage du convive i
8       convives[i].start(); // naissance du convive i
9     }
10  }
11 }
```

Résolution du problème avec un moniteur

Sortie (console)

```
1 Le philosophe 3 saisit les baguettes 3 et 4
2 Le Philosophe 3 mange
3 baguettes = [* , * , * , 3 , 3], convives = [P , P , P , M , P]
4 Le philosophe 0 saisit les baguettes 0 et 1
5 Le Philosophe 0 mange
6 baguettes = [0 , 0 , * , 3 , 3], convives = [M , P , P , M , P]
7 Le philosophe 3 libere les baguettes 3 et 4
8 Le Philosophe 3 pense
9 baguettes = [0 , 0 , * , * , *], convives = [M , P , P , P , P]
10 Le philosophe 0 libere les baguettes 0 et 1
11 Le Philosophe 0 pense
12 baguettes = [* , * , * , * , *], convives = [P , P , P , P , P]
13 Le philosophe 1 saisit les baguettes 1 et 2
14 Le Philosophe 1 mange
15 baguettes = [* , 1 , 1 , * , *], convives = [P , M , P , P , P]
16 ...
```

- baguettes : quel philosophe tient chaque baguette ;
- convives : quel philosophe pense (P) ou mange (M).

Une classe Monitor (gestionnaire de tâches)

Fichier Monitor.java

```
1 class Monitor{
2     private static final int libre = -1;
3     private int[] baguette;
4     private boolean[] mange;
5     Monitor(){
6         baguette = new int[5]; for(int i=0;i<5;i++){baguette[i] = libre;}
7         mange = new boolean[5]; for(int i=0;i<5;i++){mange[i] = false;}
8     }
9     ...}
```

L'entier `baguette[i]` est :

- libre si aucun philosophe ne la tient ;
- le numéro du philosophe qui la tient, sinon.

Le booléen `mange[i]` est :

- true si le philosophe *i* mange ;
- false, sinon.

Affichage de la situation

Fichier Monitor.java (suite)

```
1 class Monitor{...
2
3     // Affichage de l'état des baguettes et des philosophes
4     public String toString(){
5         String s = "baguettes_=-[";
6         for(int i=0;i<4;i++){
7             s += (baguette[i] != libre)?(" " + baguette[i] + ",_"):"*_,";
8         }
9         s += ((baguette[4] != libre)?(" " + baguette[4]:"*") + "]";
10
11         s += ",_convives_=-[";
12         for(int i=0;i<4;i++){
13             s += (mange[i])?("M,_"):"P,_";
14         }
15         s += (mange[4])?("M"):"P]";
16         return s;
17     }
18     ...}
```

Contrôle de la faim du philosophe

Fichier Monitor.java (fin)

```
1 class Monitor{...
2
3 // le moniteur donne faim au philosophe: chasse aux baguettes
4 public synchronized void set_affame(int i) throws InterruptedException{
5     while(baguettes[i] != libre || baguette[(i+1)%5] != libre){wait();}
6     System.out.println("Le_philosophe_" + i + "_saisit_les_baguettes_"
7         + i + "_et_" + ((i+1)%5));
8     mange[i] = true; baguette[i] = baguette[(i+1)%5] = i;
9 }
10
11 // le moniteur rend le philosophe repus: liberation des baguettes
12 public synchronized void set_repus(int i) throws InterruptedException{
13     mange[i] = false; baguette[i] = baguette[(i+1)%5] = libre;
14     System.out.println("Le_philosophe_" + i + "_libere_les_baguettes_"
15         + i + "_et_" + ((i+1)%5));
16     notifyAll(); // on reveille tout le monde: des baguettes sont libres!
17 }
18 ...}
```

Le philosophe est un thread

Fichier Philosophe.java

```
1 class Philosophe extends Thread{
2     private int i;
3     private Monitor M;
4
5     Philosophe(int i, Monitor M){this.i=i; this.M=M;}
6
7     // On redefinit le run() de Thread. Il sera invoque par start()
8     public void run(){
9         try{
10             while(true){
11                 M.set_affame(i); // le moniteur affame le philosophe: chasse aux baguettes
12                 System.out.println(Thread.currentThread().getName() + "_mange");
13                 System.out.println(M); // baguettes = [0, 0, *, 3, 3], convives = [M, P...
14                 Thread.sleep(600*(long)Math.random()); // le temps de manger...
15                 M.set_repus(i); // le philosophe est repus: liberation des baguettes
16                 System.out.println(Thread.currentThread().getName() + "_pense");
17                 System.out.println(M); // baguettes = [*, 2, 2, 3, 3], convives = [P, M...
18                 Thread.sleep(600*(long)Math.random()); // le temps de penser
19             }
20         }catch(InterruptedException e){System.out.println(e + ":_reveil_premature");}
21     }
22 }
```

Comparatif

Sans POO, les solutions à ce problème peuvent nécessiter de nombreuses lignes de code dont la technicité est parfois très importante (sémaphores Unix, threads POSIX), du genre :

```
1 if ((semid=semget(IPC_PRIVATE, NB + 1, IPC_CREAT|0600))==-1) ...
```

et recèle beaucoup d'occasions de se tromper.

La POO permet d'avoir un source simple et efficace.

Questions ?



Principe d'une *applet*

Définition

Une **applet** est un programme destiné à être vu à travers un navigateur web (appel du `.class` depuis une page HTML).

Restrictions de sécurité (*security manager*)

- accéder au système de fichier
- communiquer avec un autre site Internet que celui sur lequel l'applet a été chargé
- exécuter un autre programme que le sien
- 12 jan 2006 : faille de sécurité dans le JRE de Sun (exécution de code arbitraire) \implies mise à jour

Organisation d'une *applet* (partie HTML)

Fichier `hello_world.html`

```
1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD_HTML_4.01_Transitional//EN">
2 <html lang="en">
3 <head>
4 <meta http-equiv="Content-Type" content="text/html"; charset="iso-8859-1">
5 <title>Hello World Applet</title>
6 </head>
7 <body>
8 <div style="text-align:center;">
9 <!-- HelloWorldApplet.class est située dans le repertoire mes.applets: -->
10 <applet code="HelloWorldApplet.class" codebase="mes.applets" width="600"
11 <!-- on peut passer une option au programme (e.g. issue d'un formulaire): -->
12 <param name=taille.police value=40>
13 <!-- le message ci-dessous ne sera visible que si Java est inactif -->
14 Java est inactif: le programme ne peut etre execute.
15 </applet>
16 </div>
17 </body>
18 </html>
```

Organisation d'une applet (partie Java)

Fichier HelloWorldApplet.java

```
1 import java.awt.*, java.util.Random;
2
3 public class HelloWorldApplet extends java.applet.Applet{
4     private final int x_max = 600, y_max = 400;
5     private int x=0,y=0; // Point d'insertion depuis le coin en haut a gauche
6     private Font f = null; private String le_texte="Hello_World!";
7     private Random r = new Random();
8
9     public void init(){ // redefinition de init() qui organise la creation de l'applet:
10        setBackground(Color.white);
11        int taille = Integer.parseInt(getParameter("taille.police"));
12        if (taille <= 0){taille = 10;}
13        f = new Font("Courier",Font.BOLD,taille);
14        FontMetrics fm = getFontMetrics(f); // Informations sur la police
15        x = (x_max - fm.stringWidth(le_texte))/2; y = (y_max - fm.getHeight())/2;
16    }
17    ...}
```

Organisation d'une applet (partie Java, suite)

Fichier HelloWorldApplet.java (suite)

```
1 import java.awt.*, java.util.Random;
2
3 public class HelloWorldApplet extends java.applet.Applet{...
4
5     public void paint(Graphics screen){ // inclut le "main"
6         screen.setFont(f);
7         while(true){
8             screen.setColor(new Color((int)(r.nextDouble()*255), (int)(r.nextDouble()*255),
9                 (int)(r.nextDouble()*255)));
10            screen.drawString(le_texte,x,y);
11            try{Thread.sleep(200);}catch(InterruptedException e){}
12        }
13    }
14 }
```

On pourrait aussi redéfinir :

- `public void start()` : invoquée au chargement de la page
- `public void stop()` : invoquée lorsqu'on quitte la page
- `public void destroy()` : invoquée lors de la destruction de l'applet

Visualisation d'une *applet*

Le fichier `hello_world.html` appelle `HelloWorldApplet.class` dans Mozilla. Le texte prend une couleur 32 bits aléatoire toutes les 200 ms :



Questions ?



Conclusion

Aujourd'hui, nous avons vu :

- 1 Fichiers
- 2 Sérialisation
- 3 Threads
- 4 Applets

La séance prochaine, nous verrons :

- 4 révisions autour de String